XEROX
Office Products Division
Office Systems Business Unit

To: Ed Miller Date: June 18, 1981

From: Roy Ogus Org: SDD/SD&T Workstation Design

Subject: **Opportunities for Dandelion Cost Reduction**

Filed: [Iris]<Ogus>memo>DLionCostRed.press

Introduction

This memo explores some of the opportunities that exist for reducing the manufacturing cost of the current Dandelion processor. All the changes discussed here relate only to the Dandelion electronics; cost reductions through reducing the cost of the housing, cables, peripherals, and power supply, are not covered in this memo.

A significant reduction in cost will occur if an entire logic module can be dispensed with in the system. The removal of a gate here and a flip-flop there is usually not meaningful unless the sum of these small logic savings add up such that an entire function can be moved from one module to another, potentially eliminating the module.

The changes discussed below are divided into two broad categories, viz. straightforward changes, i.e. those that can be made without a large redevelopment effort, and longer term changes, i.e. those that require a substantial development or tools effort, or are dependant on components not yet available.

Straightforward Cost Reduction Opportunities

Straightforward opportunities for cost reduction in the Dandelion will be available either when more highly integrated memory components become available at a competitive price, or when some of the optional functions are removed from the systems. This sections explores these possibilities.

Memory system

a) 64K Dynamic RAMs

The first potential cost reduction technique is to convert the main memory RAM component from the current 16K dynamic RAM to the 64K dynamic RAM. When this occurs, there is the possibility of implementing up to 256K words on the Memory Controller, thus dispensing with the Storage module on systems which do not require more than 256K words. (It is expected than the 8000 processors will not need more than this for some time.) Even though an entire module can be removed from the system, the system savings still depends on the cost of the 64K RAM chip. At current prices, it is still more expensive to implement the memory system with 64K chips rather than 16K chips. The point at which there is a cost benefit by using 64K chips can be computed as follows.

Let C_{16} and C_{64} be the cost of 16K and 64K chips respectively. The materials cost of the Dandelion memory modules can be expressed by

$$Cost(PWB) + Cost(Burn\ In) + Cost(RAM\ parts) + Cost(other\ parts)$$

$$Cost(PWB) + Cost(Burn\ In) + C_{xx}*N_{xx} + Cost(other\ parts), \quad (1)$$

where "xx" is either "16" or "64", and N_{xx} is the number of type "xx" RAM chips used. [Source: *OSBU Program Management Book (OSBU-PMB),* section on *Manufacturing and Distribution.* Note that Labor and Overhead costs are not included.]

Let *MCC-16* and *MSC-16* denote the 16K-based Memory controller and Storage modules respectively. Let *Cost (16)* denote the cost of the 16K-based memory system containing 192K words.

Thus, from (1) and *OSBU-PMB* (using the base costs),

$$\begin{aligned} Cost(MCC\text{-}16) &= \$94 + \$6 + \$91 + 88*C_{16} \\ &= \$191 + 88*C_{16} \end{aligned}$$

and,

$$\begin{aligned} Cost(MSC\text{-}16) &= \$72 + \$2 + \$46 + 176*C_{16} \\ &= \$120 + 176*C_{16} \end{aligned}$$

Thus, from (1)

$$\begin{aligned} Cost(16) &= Cost(MCC\text{-}16) + Cost(MSC\text{-}16) \\ &= 311 + 264*C_{16} \end{aligned} \quad (2)$$

A 64K-based memory controller, *MCC-64*, containing 192K words, will have a cost, from (1) and *OSBU-PMB* (using the base costs), of

$$\begin{aligned} Cost(64) &= Cost(MCC\text{-}64) \\ &= \$94 + \$6 + \$91 + 66*C_{64} \\ &= \$191 + 66*C_{64} \end{aligned} \quad (3)$$

The use of 64K memory chips becomes more economical when the price of the 64K chips drops relative to the 16K devices such that the cost of the memory sytem implemented by 64K devices is less than the cost of the system implemented by 64K devices. Thus, using (2) and (3),

$$\begin{aligned} Cost(64) &< Cost(16) \\ \$191 + 66*C_{64} &< 311 + 264*C_{16} \\ C_{64} &< 1.82 + 4*C_{16} \end{aligned} \quad (4)$$

The following table indicates the relative costs of the 64K and 16K devices such that the 64K-based memory system containing 192K words becomes cheaper than the 16K-based implementation:

16K cost (C_{16}) [$]	*64K cost (C_{64}) [$]*
1	5.82
2	9.82
3	13.82
4	17.82

Thus, for example, if the cost of a 16K chip is $2, then the 64K chip must cost $9.82 or less for the 64K-based implementation to be cheaper.

Actually, the costs not considered here, viz. the labor and overhead costs, will change the values in the table above. For example, the *OSBU-PMB* specifies base costs of the 16K and 64K chips as $2.87 and $14.00, respectively. According to the table above, for a 16K cost of $2.87, the break-even 64K price is $13.35, indicating that, considering materials costs alone, the 64K is still not economical. However, it is shown that when labor and overhead costs are included, the 64K-based implementation will be less costly.

To get an idea of the amount of UMC reduction possible, we can consider the difference in memory costs. From (2) and (3) the difference,

$$Cost(16) - Cost(64) = \quad 120 + 264*C_{16} - 66*C_{64} \qquad (5)$$

Assume the cost of the 16K device is $2.87. If the cost of the 64K device is $13.35 then from (5) the memory system will cost the same. If the price of the 64K chip goes down to $10, then the 64K-based system will cost $218 less.

b) Removal of Error Correction Logic

Another method to reduce the memory system cost is to remove the error-correction logic. The simple parity checking would remain, so that all single errors would be trapped. Note that this cost reduction method should only be considered when more data is available on the memory system error characteristics, indicating a low enough probability of single errors. Only the case using the 64K memory devices is considered here.

Removing the error correction logic from the memory system will discard 5 memory chips per memory bank. In addition, 9 other miscellaneous chips can be dispensed with in the memory system. Thus, for the 192K memory implemented in 64K chips (three banks), the cost savings will be

$$9 + 15*C_{64} \qquad (5)$$

For $10 64K chips the savings will be $159, while $8 64K chips will save $129.

Central Processor

The RAM-implemented control store (48 chips) comprises 30% of the total number of CP integrated circuits. The control store is 4K x 48 implemented using forty-eight 2147L 4Kx1 RAM chips. Possible ways to reduce the control store cost are through the use of higher density RAMs or through conversion to Prom which can be found in more dense packages.

a) Higher density RAMs

The current control store RAMs are 4K bit RAM chips (2147L), requiring 48 chips to implement the array. 16K bit RAM chips exist which are configured as 4K x 4 bits. The use of the 2168 chips would cut down the number of chips by a factor of 4. If the cost of the 2168 chips is less than four times the cost of the 2147 chips then this change would be cost-efective. Current costs of the 2147 and 2168 chips are approximately $5.25 and $30, respectively, however the cost of the 2168 parts is expected to drop below $20 in 1982.

This change has no system implications and could be easily incorporated into the CP design. However, the tasks of qualification of the 2168 parts, and the determination of second sources for the parts, would have to be carried out.

b) Prom Control Store

Another potential cost savings could be gained if the control store would be implemented in Prom instead of the less-dense RAM. 4Kx4 and 4Kx8 proms will be available from Monolithic Memories, Inc. (part numers 63RA1641 and 63RA1681) within about a year, which would cut down the number of components from 48 to 12 or 6 components respectively. The acual cost savings (if any) would have to be computed using the relative costs of the Prom and RAM chips (the Prom prices are not yet available).

It should be pointed out, however, that converting to Prom control store in the CP will incur other additional costs, as well as having a large impact on the system microcode and IOP software. Each Prom would be a distinct part which has to be produced and inventoried, adding to the number of different components in the system. If the part is not qualified (as will almost certainly be the case), a qualification program would have to be carried out with its attendant costs.

The microcode could not be changed dynamically, precluding the use of the same Dandelion for more than one configuration (e.g. as a Workstation or a Raven Printer Server), unless the microcode for all the configurations could be stored together in the control store. Currently this is not possible due to lack of space.

Subsequent changes to the microcode would be tedious and costly to implement since this would involve retrofitting substantial numbers of machines in the field.

In addition, there are large system implications. The booting process utilizes the writeability in several ways. Initial diagnostics are loaded into the control store and executed, and device and system initialization procedures are executed before the standard microcode is loaded. The standard microcode is close to filling the control store. There are also stand alone diagnostics which utilize writeable control store.

If the control store is changed from RAM to Prom, then these functions can only be executed from microcode which is always present within the control store. The 4K control store size prohibits these functions from being accomplished in the Prom Dandelion in their current form. There are two possible options. One is to have the Prom Dandelion contain more than 4K of control store. The second is to develop diagnostic and initialization strategies which would use an absolute minimum of control store space. It is probable that 4K is still not large enough. Consideration of the diagnostic strategy changes will require discussions with people from the diagnostic area, and probably people from manufacturing because of the new limitations during assembly and test. Changing the booting strategy would involve large changes to the microcode, as well as to the boot code (microcode and IOP software), including the EProm code.

c) CMOS Control Store RAMs

Another type of cost reduction is to reduce life cycle costs by reducing the amount off power consumed by the system. This will improve reliability and result in cost savings through a lowered failure rate of the system. In addition, this will alleviate the current burden on the power supply. It is known that the use of 64K memory components (especially when the 64K-based MSC is implemented) will impose a large burden on the +5V power supply, perhaps even exceeding the designed output in some cases (e.g. using a highly populated XMSC module).

One way to reduce the +5V current significanrly is to use CMOS control store RAM components. Hitachi makes a CMOS RAM component, the HM6147, which is pin-for-pin compatible with the 2147L 4Kx1 RAM currently used in the control store. The 2147L implemented control store uses 6.72 amps from the +5V power supply, and dissipates 36 watts of power. The comparable numbers for the HM6147 are 0.72 amps and 3.6 watts, an order of magnitude reduction.

The HM6147 CMOS RAM is currently a single-sourced component, but there are plans for Motorola (and perhaps Intel) to second source the component. The component would also have to be qualified before it can be considered a viable replacement for the 2147.

Options Module

Raven and RS232C controller

Other savings can be realized if certain optional functions are removed from the system. The cost saving would of course only be applicable in configurations not requiring the particular function. Two such optional functions are the Raven controller and the RS232C/RS366 controller. Removing the Raven control would save about 20 equivalent chips. The corresponding chip savings for the RS232C/RS366 controller is about 30 equivalent chips. The cost savings would be more than a $1-a-chip since there are several LSI components in the two controllers. The Raven controller would save approximately $30, while the RS232C controller could save about $60.

IOP Module

Removal of Floppy Disk Controller

A savings of about 28 equivalent chips (approximitely $40) can be achieved if the floppy disk controller is removed from the IOP. There would be an additional savings of about $300 for the removal of the drive. This change can probably only be contemplated for a workstation machine, which has Etherbooting implemented.

There are other implications in removing the floppy disk. The diagnostics are run from the floppy disk, and cases where the CP does not function, the machine can still be diagnosed from the IOP. This would not be possible if there was no floppy disk, since the diagnostics would have to be fetched from the Ethernet, requiring the CP to be functional. In addition, software distribution is currently achieved using floppy disks. This would have to be redesigned to allow distribution over the Ethernet. Initialization of a server machine on a newtwork with no other servers can, of course, not do without the floppy disk

Conversion to CMOS Time-of-Day circuit in Maintenance Panel

The original Time-of-Day (TOD) circuit which was designed for the Dandelion was implemented in CMOS logic, stored entirely on the Maintenance Panel module. On recommendation of the ED/Parts group this design was abandoned in favor of a completely TTL design. This had the effect of requiring additional space on the IOP module to house the most of the TOD circuitry. If the original TOD circuit was re-used, then 12 chips on the IOP could be dispensed with. There still might have to be qualification activity on some of the parts. This savings is small, and also does not free up enough room on the IOP to allow moving any other function. This change is documented here for the record.

Conclusions

This section summarizes the potential savings described above. Note that some of the techniques have little or no effect on the system operation, while others have a large impact. This impact should first be evaluated before considering using the cost-reduction technique.

64K Dynamic RAMs. This technique is clearly one that should be implemented. The economics are such that the 64K-based system is on the verge of being (or is already) cost-effective. Considerations still to be resolved are the availability of 64K parts, and the qualification of suitable vendors for supply.

Removal of Error Correction Logic. More data on and analysis of 64K memory failure rates in the system is needed before this technique should be considered.

Higher density RAM Control Store. This change is attractive since it frees up 36 chip locations on the CP card, an is a relatively simple change to implement. However, there is still some risk in that the 2168 parts are still not available in production quantities. It is expected that this will not be a problem in 1982.

Prom Control Store. This change should be considered with great caution. The appropriate Prom chips are not yet available, and thus not qualified or second-sourced. It is not clear yet whether Prom usage will result in a cost savings (or at least initially). But most importantly, this change has a large impact on the system operation. The ramifications of a Prom control store on booting, diagnostics operation, and handling of system changes, should be studied carefully before deciding to implement the change.

CMOS Control Store RAMs. It is difficult to put a value on the cost savings of this change, since the savings will be in life-cycle costs due to increased reliability (hopefully). Other benefits of this change are to reduce the load on the +5V power supply, thus increasing the overload margin when the 64K-based memory is used.

Removal of Raven and/or RS232C Controller. This savings can only be realized in machines that can dispense with these functions. Other benefits will occur if other savings together with these result in the entire Options card being able to be dispensed with.

Removal of Floppy Disk Controller. This savings can be achieved in machines that do not need the floppy disk. This will depend on Etherbooting being implemented, and the corresponding software changes to initialize the rigid disk from the Ethernet. There are also Field implications, in that diagnostics, and distribution of software will have to be redesigned.

CMOS Time-of-Day circuit. A relatively small savings is achieved using this technique. It might have the added benefit of isolating the TOD from the system in a better manner than is now done, perhaps reducing the interference problems in the TOD circuitry.

Thus, the recommended cost reductions are the use of 64K memory chips, and the 4Kx4 control store chips. In addition, the Raven and RS232C controllers may be dispensed with in certain configurations.

A significant savings will be achieved if the Dandelion can be reduced to a 4-board machine (Memory Control, CP, IOP, and HSIO). Implementing the 64K-based memory system, using the 4Kx4 control store chips, and removing the RS232C and Raven controllers, still does not allow the Options card to be removed since the Ethernet controller (almost 100 equivalent chips currently) has to placed on the other boards, which do not obviously have the space. More work needs to be done to determine how to remove the fifth board (if that is possible).

Longer Term Cost Reduction Opportunities

The changes described in this section are those which are not immediately feasible. This may be due to a substantial development effort being required, together with the implementation of new design tools. Alternatively, the change may be dependent on usage of vendor components which are not yet available.

a) Ethernet controller

Intel and other vendors are currently developing LSI components specially designed to implement Ethernet controllers for LSI microprocessor systems. These components are more than 18 months from being in production.

It is not apparent to me that the Intel Ethernet controller chip (82D2-E) would allow any cost reduction in the Dandelion Ethernet controller. It is speciafically designed to interface to an 8086-like bus structure which is not compatible with the Dandelion I/O controller bus structure. A reasonable amount of logic would have to be implemented to convert between the two bus structure, and this could offset any potential savings through the use of the LSI controller. The 82D2-E interfaces to the processor through memory, and thus a two-ported memory system would have to be added to the controller for communications to the CP.

The Intel Ethernet Serial Interface (ESI) chip, however, appears to fit in more easily with the current controller structure. The ESI integrates the front-end controller functions such as transceiver interfacing, and data stream encoding and decoding (phase-lock loop). Using the ESI chip could save approxiately 14 chips and about 60 discretes in the Ethernet controller.

b) Gate Array usage

This section takes a first look at the potential for implementing parts of the Dandelion CP module using gate-array techniques. The available Texas Instruments gate array components were used as the basis the study.

Assuming that as much of the CP was converted to gate array implementation, then the Dandelion CP could be implemented with the following components:

?	Control-Store chips (depends on whether RAM or Prom)
7	Gate Array Chips
11	RAM Memory Chips (4-256x4, 7-16x4)
4	2901 ALU Chips
15	MSI Chips

In some cases, it may not make sense to implement the section of random logic in gate arrays, since it still may be cheaper to implement the random logic. However, this analysis serves to illustrate the largest savings in chips that would be possible.

The details of the gate array chips are as follows:

LRot-StkP-IB-NibByte
replaces 29 IC's
contains 1000 gates
has 67 I/O signals

NIA0..7
replaces 7+ IC's
contains 294 gates
has 64 I/O signals

NIA8..11
replaces 9+ IC's
contains 192 gates
has 66 I/O signals

FDecodes
replaces 7+ IC's
contains 160 gates
has 64 I/O signals

Error-Kern-Task
replaces 9+ IC's
contains 337 gates
has 71 I/O signals

Carry-Shift-DispBr
replaces 7+ IC's
contains 156 gates
has 52 I/O signals

Clock-Mar-Misc
replaces 9+ IC's
contains 189 gates
has 67 I/O signals

Appendix A contains the full details of what logic in the CP is replaced by the various gate array chips

Appendix A: Gate Array Component Details

Following are the details of the gate arrays referenced in the memo. The numbers in braces indicate the number of gate array gates required for each chip which is replaced. The TI gate arrays were used as basis of this study. The information on the gate array implementations was provided by Don Charnley.

LRot-StkP-IB-NibByte

S257	{64}
25S09	{45}
LS283	{60}
25S09	{45}
S240	{9}
S373	{81}
S373	{81}
S374	{65}
S260 (1/2)	{7}
LS374 (2/4)	{33}
F93453	{20}
F93453	{20}
25S10	{40}
25S10	{40}
25S10	{40}
25S10	{40}
S241	{16}
S257	{64}
S257	{64}
S257	{64}
S241 (4/8)	{8}
S241 (4/8)	{8}
S138 [3/8]	{10}
S138 [1/8]	{4}
S138 [2/8]	{7}
S138[5/8]	{16}
S00 (1/4)	{1}
S00 (1/4)	{1}
S20 (1/2)	{1}
S20 (1/2)	{1}
F93453 (1/4)	{10}
S10 (1/3)	{1}
S00 (1/4)	{1}
S10 (1/3)	{1}
S00 (1/4)	{1}
S138 [3/8]	{10}
S374 [4/8]	{33}
LS32 (1/4)	{3}
S02 (1/4)	{3}

Inputs:

X.0..X.15
Y:0..Y.15
fS.0..fS.3
fX.0..fX.3
fY.0..fY.3
pfZ.0..pfZ.3
pfS.2, pSE, AlwaysClk, WaitClk, ppCLK
Wait, AllowMDR←, MesaInt

Outputs:

IB.0..IB.7
IBEmptyErr, EKErr.0', EKErr.1'

NIA0..7

LS158	{13}
LS158	{13}
S374	{65}
S374	{65}
LS374 (8/8)	{65}
LS244 (3/4)	{7}
25S09	{45}
25S09	{45}

Inputs:

INIA.0..INIA.7
MesaInt, IBPtr.1
IB.0..IB.3
RefillIntc2, GoodIBDispc2, EKTrapc2
AlwaysClk, SwTAddr
IOPData.0..IOPData.7
TPC.0'..TPC.7'
Swc2, WrTPCHigh'

Outputs:

Nt.0..Nt.2
pNIA.0'..pNIA.7'
NIAX.0'..NIAX.7'
NIA.0'..NIA.7'

NIA8..11

S64)	{6}
S64)	{6}
S64)	{6}
S64)	{6}
S00 (2/4))	{2}
S10 (2/3))	{2}
S374)	{65}
LS244)	{9}
25S09)	{45}
LS32 (4/4))	{12}
S374 (4/8))	{33}

Inputs:

pLink.0'..pLink3'
DispBr.0'..DispBr.3A',DispBr.3B', MarPgCross'
INIA.8..INIA.11
IB.4..IB.7
TC.0..TC.3
Swc3', EKTrapc2', GoodIBDispc2
AlwaysClk, SwTAddr
IOPData.4..IOPData.7
TPC.8'..TPC.11'
Swc2
fX.0, NIAX.7'

Outputs:

pTC.0..pTC.3
pNIA.8'..pNIA.11'
NIAX.8'..NIAX.11'
TCX.0..TCX.3
Link.0'..Link3'
NIA.8'..NIA.11'

FDecodes

S138 (8/8)	{25}
S138 (8/8)	{25}
S138 (8/8)	{25}
S138 (4/8)	{13}
S138 (5/8)	{16}
S138 (6/8)	{19}
S138 (5/8)	{16}
S138 (3/8)	{10}
S00 (3/4)	{3}
LS32 (4/4)	{8}

Inputs:

fY.0, fY.1, fY.2, fY.3, fX.0, fX.1, fX.2, fX.3, fZ.0, fZ.1, fZ.2, fZ.3, fS.0, fS.1, fS.2, fS.3

Outputs:

ExitKernel', EnterKernel', ClrIntErr', MesaIntRq', CycleY'
MapRefY', RefreshY', PushY, ClrDPReq', ClrIOPReq', ClrXReq', ClrKReq'
RH←', Shift', CycleX', CIN←pcl6X', MapRefX'
RefreshZ', CIN←pcl6Z', AltUAddr'
IOPOData←', IOPCtl←', KOData←', KCtl←', EOData←', EICtl←', DCtlFifo←', DCtl←'
DBorder←', PCtl←', MCtl←', IOutSp1←', EOCtl←', KCmd←', IOutSp4←', POData←'
←EIData', ←EIStatus', ←KIData', ←KStatus, KStrobe, ←MStatus', ←KTest', EStrobe
←IOPIData', ←IOPStatus', ←ErrIBStkp', ←RH'

Error-Kern-Task

F93453	{20}
LS374 (8/8)	{65}
S241 (4/8)	{16}
F93453	{20}
S374 (2/8)	{17}
S02 (1/4)	{2}
S374 (3/8)	{25}
F93453	{30}
F93427	{20}
F93453	{30}
LS374 (6/8)	{49}
LS374 (5/8)	{41}
S08 (1/4)	{2}

Inputs:

IOPData.0..IOPData.2, WrTPCHigh', SwTAddr'
EORound, Click.0, Click.1, Click.2
DPReq', IOPReq', EReq', KReq', RefReq', RefReq', KernReq'
SwTAddr, Cycle2, Wait
CSPar.0..CSPar.5, MesaIntRq', ClrIntErr'
MemErrc3, Pt=Emu, WaitClk, AlwaysClk

IBEmptyErr, Ct=Emu, Cycle1
YH.0, YH.1, MapRef
ExitKernel', EnterKernel', IOPWait, Cin←pc16

Outputs:

Nt.0, Nt.1, Nt.2, Nt=Emu
Ct.0, Ct.1, Ct.2, Ct=Equ
Pt.0, Pt.1, Pt.2, Pt=Emu
Swc2, Swc2', Swc3, Swc3'
MesaInt, EmuMemErr, CSParErr, VirtAddrErr
pEKT, pEKT', pEK0', pEK1'
EKTrapc2', EKTrapc2, EKErr.0', EKErr.1'
KernReq', pc16'

Carry-Shift-DispBr

LS158 (2/4)	{7}
LS32 (4/4)	{12}
S151	{26}
S151	{26}
S151	{26}
S182	{24}
S86 (1/4)	{6}
S04 (1/6)	{1}
S253	{24}
S38 (4/4)	{4}

Inputs:

fY.0, fY.1, fY.2, fY.3
X.0, X.4, X.8, X.12..X.15
fS.0, fS.1
Y.12..Y.15
F.0, Cycle2, YIODisp.0, YIODisp.1, MesaInt, NibCarry, OVR
CIN-SE
G4-7', G8-11', G12-15'
P4-7', P8-11', P12-15'
aF1.2, FZero, Carry, R.15, Q.0, R.0, aD.1, Shift', aD.0
CIN←pc16, pc16', Cin←pc16X', CIN-SE-wrSU'

Outputs:

DispBr.0', DispBr.1', DispBr.2', DispBr.3'
CIN0-3, PageCarry, PageCross, FNZero

Clock-Mar-Misc

S00 (4/4	{4}
S00 (2/4	{2}
S02 (4/4	{12}
S04 (6/6	{6}
S04 (1/6	{1}
S08 (1/4	{2}
S10 (1/3	{1}
S51 (2/2	{4}
LS32 (1/4	{3}
S260 (1/2	{14}
S374 (3/8	{25}
S00 (2/4	{2}
S04 (2/6	{2}
S374 (2/8	{17}

S08 (1/4 {2}
25S09 {45}
25S09 {45}
S08 (1/4 {2}

Inputs:

ppCLK, Cycle1', Cycle2', Cycle3'
pAlwaysCLK, TCWaitc1', CIN-SE-wrSU, NIAX.7', RH←'
IBFront←, WrTPCLow, IB←', IOPWait, Disp-Proc', MAR←'
YH.4..YH.7
pmem, MAR←, PageCross, IBEmptyErr
prA.0..prA.3
paS.0, paS.1, paS.2, paF.1, paF.2, pMAR←'

Outputs:

Cycle1, Cycle2, Cycle3
AlwaysClk-a, AlwaysClk-b, AlwaysClk-c, AlwaysClk-d
WriteTC', C2Clk
WaitClk, WriteSU', WriteLink', WriteRH', WrIBFront, WriteTPC', WriteIB
TCWaitc1', Wait
pMAR←', MAR←', MAR←, MarPgCross', IBEmptyErr', pAllowMDR←', AllowMDR←
SUAddr.0..SUAddr.3
aSh.1, aSh.2, aFh.1, aFh.2, paSh.0

c: Distribution